August 1981

8298 Integrated E^2 Controller

Kevin Corbett, Larry Palley and Randy Battat
Special Products Division Applications Engineering

INTRODUCTION

The Intel 8298 is a special function peripheral device designed to interface Intel's full line of Electrically Erasable PROM memories to Intel microprocessor systems. It also enables the interface of E^2PROM devices to any system using an 8-bit I/O data bus. The 8298 is used to supervise the writing, erasing and reading of up to 8 2816 E^2PROMs. It is used because the write and erase cycles of E^2PROMs involve significant amounts of time (on the order of 10 ms each). A large portion of CPU time would be required to supervise these operations if no interface element existed. Adding the 8298 enables the supervision of these operations off-line, thereby freeing up the CPU and effectively increasing the throughput of the total system.

The 8298 is designed to enable a choice of hardware configurations. The use of the 8298 in the INDIRECT mode minimizes the hardware required to interface to the E^2PROMs, but sacrifices the ability to read and write the E^2PROMs directly. Adding extra hardware enables the 8298 to be used in the DIRECT mode, allowing read access at full system speed.

The goal of this application note is to facilitate simple and easy implementation of the 8298 and E^2PROMs by hardware and software engineers in their systems. Described herein are the 8298 and design configurations in which it can be used. The material is structured so that this application note can be a handy reference guide to the 8298 commands and to the two hardware configurations presented.

UPI-41A Base

The 8298 E^2 interface is based on the UPI-41A Universal Peripheral Interface. Special firmware has been written so that implementation of the E^2 interface can be undertaken in the simplest and most efficient manner. All timing signals for the 8298 are, therefore, identical to those of the UPI-41A. The use of the UPI-41A base also makes the 8298 fully compatible with all Intel microprocessors.

8298 HARDWARE

A pinout diagram of the 8298 and 8243 I/O expansion module are shown in Figure 1. The following section describes how these devices are hard-wired into the system. Each of the pin functions are described.

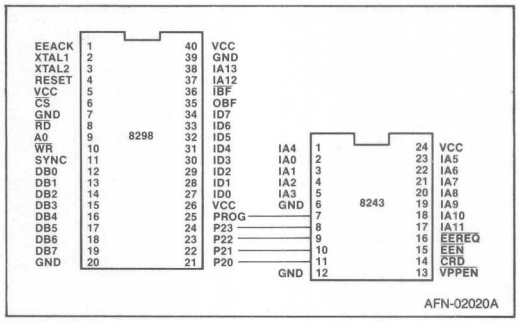


Figure 1. Pinouts of 8298 and 8243

System Interface

Interfacing the 8298 to the system is easily achieved. The 8298 is wired directly to the system bus as an I/O port with the data bus connected to the DBO-7 lines. All command, status, and data bytes are passed on this interface. The 8298 is addressed as a port using external I/O address decode hardware to enable the chip select ($\overline{CS}$) line. Only the A0 line is connected directly (or through a buffer) to the least significant I/O address line.

The A0 line selects between command and data input registers on writes to the 8298 and selects between the status and data output register during reads. By strobing the $\overline{RD}$ line of the 8298, data is enabled onto the data bus for reads. Writing data to the data bus occurs on the rising edge of the $\overline{WR}$ signal.

Two external pins are available for use as interrupts. The $\overline{IBF}$ line, when inactive, indicates that the 8298 is waiting for an additional command or data byte. The OBF line indicates to the processor that the data output register is full. The processor must service the 8298 during reads (when OBF active) by accessing data from the data output port. This allows the 8298 to continue reading the contents of the E^2PROMs. The $\overline{IBF}$, when activated during writes, indicates to the processor that it must halt data transfers to the 8298. The 8298 must write the contents of the data buffer to the E^2PROMS before the CPU continues to transfer data.

E^2PROM Interface

An array of 8 2816 E^2PROMS has 16K bytes of memory. The 8298 alone cannot supply the necessary 14 address lines for this array, so the 8243 I/O expansion module is used. The 8243 provides the twelve low-order address bits. The 8-bit internal data bus, and extra control pins are available from the 8298 directly. Adding the 8243 expander is implemented by connecting the

PROG and the P20-P23 lines of the two devices together. All necessary control and addressing data for the 8243 are passed from the 8298 through these lines.

Several control signals are necessary to read, write, and erase the E^2PROMS. These signals, supplied by the 8298 and 8243, are described individually below:

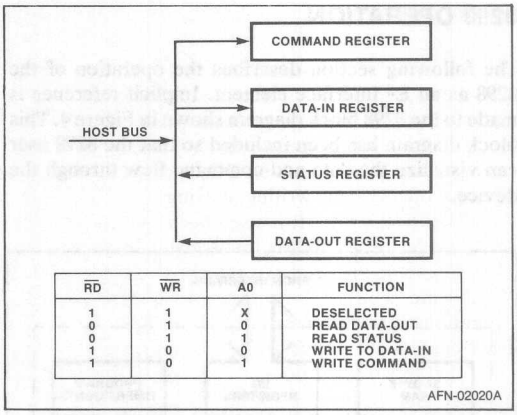
1. $\overline{EEN}$—E^2 enable is generated to access the E^2 array.
2. $\overline{CRD}$—Controller read is generated to indicate to the E^2PROM array that data is being read by the 8298.
3. $\overline{VPPEN}$—Programming pulse enable is generated to indicate that the programming pulse for a write should be applied to the E^2PROM array.
4. $\overline{EEREQ}$—E^2 request is generated to request access to the E^2PROM array in the direct configuration.
5. EEACK—E^2 acknowledge indicates that the E^2PROM array can be accessed by the 8298 in the direct configuration. It is generated by arbitration circuitry.

8298 Registers

The 8298 has four registers which the host CPU can access from the system data bus. Figure 2 shows the four registers and the method for selecting one of them. $\overline{RD}$, $\overline{WR}$, and A0 are all signals which are generated by the host CPU to perform register operations.

Command and Data Registers

The first byte of every command goes to the 8298 command register. The byte will be interpreted to determine the type of command and if additional bytes are required for the command. If a command has more than one byte, the additional bytes are sent to the data-in register. All data which is to be sent to the 8298 is written to the 8298 data-in register.



$\overline{RD}$	$\overline{WR}$	A0	FUNCTION
1	1	X	DESELECTED
0	1	0	READ DATA-OUT
0	1	1	READ STATUS
1	0	0	WRITE TO DATA-IN
1	0	1	WRITE COMMAND

Figure 2. 8298 Ports and Selection

When the 8298 performs a read operation, the data read by the host CPU is accessed through the data-out register.

Status Register

The 8298 status register contains a byte of information which describes the current state of the 8298. The host reads this register to determine if the 8298 is waiting for bytes of command or data, has received an illegal command, or is currently busy executing a command. Figure 3 and Table 1 describe the bit definition and bit interpretation of the status byte.

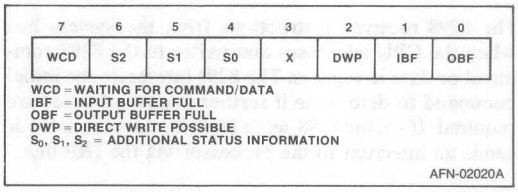


Figure 3. 8298 Status Word Bit Definition

Table 1. Status Definition

WCD	S2	S1	S0	OBF	DESCRIPTION
1	1	1	1	0	Waiting for Command
0	0	0	0	0	Executing Command
1	1	1	0	0	Illegal Command
0	0	X	X	1*	Illegal Command During Write
1	1	1	1	1	Read Command Complete
0	0	0	0	1	Read Data Ready
1	0	0	0	0	Waiting for Data Byte 1
1	0	0	1	0	Waiting for Data Byte 2
1	0	1	0	0	Waiting for Data Byte 3
1	0	1	1	0	Waiting for Data Byte 4
0	1	0	1	0	Write in Process

*Output buffer contains 10101010.

8298 OPERATION

The following section describes the operation of the 8298 as an E^2 interface element. Implicit reference is made to the 8298 block diagram shown in Figure 4. This block diagram has been included so that the 8298 user can visualize the data and command flow through the device.

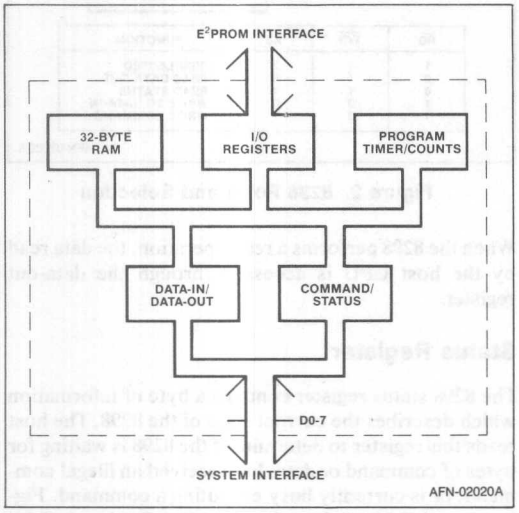


Figure 4. 8298 Block Diagram

The 8298 receives instructions from the system bus when the CPU addresses and writes to the 8298 command or data-in register. The 8298 interprets the initial command to determine if further command bytes are required. If so, the 8298 sets a bit in its status register or sends an interrupt to the processor via the $\overline{IBF}$ line.

After a full command string has been received, the 8298 determines whether the command is a read, write, erase or utility instruction. If it is a read or write command, data transfer begins. The CPU writes or reads data from the 8298 through the data input and output registers. This data is transferred directly to or from the E^2PROMS or buffered temporarily in 32 bytes of internal RAM, in the case of a multiple or series write. The 8298 indicates to the CPU when a data transfer is complete.

Series or multiple write transfers utilize the 32-byte internal data RAM buffer to speed system throughput. Using this buffer enables the CPU to write up to 32 bytes of data at system speed. The CPU uses this buffer by sending special write commands.

For commands where multiple bytes are written, the CPU sends a series of data bytes to the input register from which they are taken and stored in the buffer. When the buffer becomes full, the host processor can determine, via the status register, that data transfer to the 8298 must be temporarily suspended. The data in the full buffer is then written to the E^2PROMs, while the processor is freed for other processing. The 8298 notifies the CPU that data transfer can continue when the buffer is empty, with the IBF status or interrupt.

Read commands to the 8298 are all processed immediately. After receiving a read command, the 8298 accesses the E^2PROM array at the appropriate address and transfers that data byte directly to the data-out register. The host CPU is notified via the OBF status or interrupt to read this byte.

Utility commands are used to set up the 8298 configuration (DIRECT or INDIRECT), to initialize the timer associated with writing and erasing the E^2PROMs, to abort current commands, and to determine the address or data last written or read from the E^2PROMs.

CPU/8298 Communication

CPU communication with the 8298 can be implemented through software polling routines or through the use of interrupts.

In the polling routines, the CPU reads the contents of the 8298 status register at regular intervals to determine if data transmission is possible. When the status of the 8298 indicates that it is ready to accept a command or data byte or that it has data ready to be transmitted, the CPU can branch to a routine which performs these functions. The use of polling routine is easy to implement in software and requires no hardware overhead. Polling, however, takes a significant amount of CPU time. This time could be put to more efficient use.

The use of 8298 interrupts in place of polling routines eliminates wasted CPU time. The CPU can send a command string to the 8298 and let the 8298 execute it while it is devoting time to other tasks. When the 8298 completes command execution, it can interrupt the CPU. The CPU can then service the interrupt by returning to an 8298 communication routine.

A sequence in which a command string is transmitted to the 8298 is flowcharted in Figure 5. In this flowchart are two conditional states where WCD set and IBF clear are tested. A polling routine would be used to read the status register of the 8298, test these two bits, and loop

back if they are not set. An interrupt routine would be used to indicate the same state of these bits, but while the CPU is processing another task.

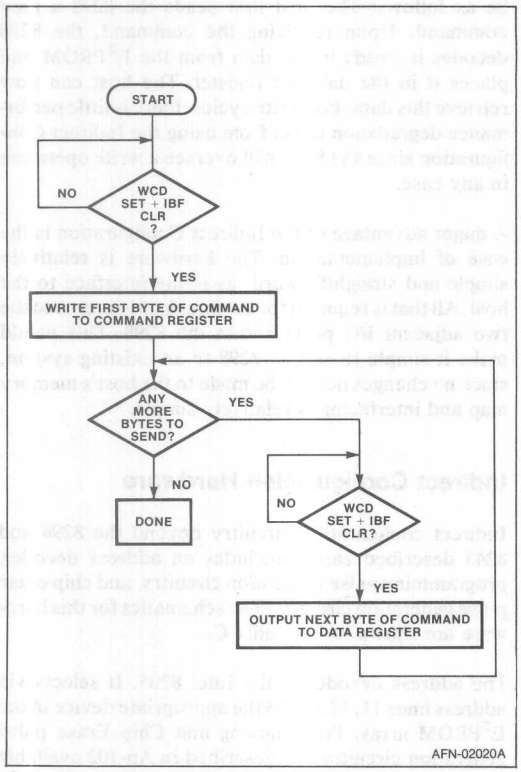


Figure 5. Flowchart for Command Transmission

Programming examples of this flowchart are shown in Figures 6 and 7. These routines are written in 8085 assembly language and are easily upgraded to 8086 code. The first set of routines is the actual module used to service the 8298. These routines are callable from a main program. The second set of routines are an example of the mulitple write command as it may exist in a main program. In this module the last statement is a call to the 8298 service routine in Figure 6 to write 10 data bytes to 10 different E^2PROM addresses.

SNDCMD and SNDDBX are subroutines used to send a command of one or more bytes to the 8298. They read the number of bytes required for the command from a command buffer (which the user has set up) and send them one at a time to the 8298. Each transmission monitors the status of the 8298 before taking place. CHKSTS is the polling used for this purpose.

This same module can be used in an interrupt-driven mode. The processor would initially call the SNDCMD routine to transmit the first command byte to the 8298. After returning to the other tasks, the CPU would later be interrupted by $\overline{IBF}$ and would send the next byte via the SNDDBX routine. Each ensuing interrupt issued by the 8298 would call the SNDDBX routine until all command bytes have been sent. The routine CHKSTS would not be needed when using interrupts. Significant savings in CPU time would result.

```
SNDCMD:  PUSH   PSW
         PUSH   B
         PUSH   D              ;SAVE REGISTERS

         LXI    H,CMDBUF       ;GET BYTE COUNT
         MOV    D,M
         INX    H
         MOV    E,M

         INX    H

         MVI    C,WCD
         CALL   CHKSTS

         MOV    A,M
         OUT    CMND           ;SEND COMMAND BYTE

         MOV    A,D
         ORA    A
         JNZ    DECD           ;TEST BYTE COUNT

         MOV    A,E
         ORA    A
         JZ     ENDSND         ;IF ZERO, DONE

DECD:    DCX    D
         MVI    C,DB1STS

SNDDBX:  INX    H              ;IF NOT ZERO, SEND

         CALL   CHKSTS         ;MORE DATA BYTES

         MOV    A,M
         OUT    DATA

         MOV    A,D
         ORA    A
         JNZ    DECD2

         MOV    A,E
         ORA    A
         JZ     ENDSND

DECD2:   DCX    D              ;CHANGE EXPECTED STATUS
         MOV    A,C            ;VALUE
         ADI    10H
         MOV    C,A
         JMP    SNDDBX

ENDSND:  POP    D
         POP    B
         POP    PSW            ;RESTORE REGISTERS
         RET

CHKSTS:  IN     STS            ;POLLING ROUTINE
         MOV    B,A
         IN     STS
         CMP    B
         JNZ    CHKSTS
         CMP    C
         JNZ    CHKSTS
         RET
```

AFN-02020A

Figure 6. 8298 Service Routine

```
; CONSTANTS
; PORTS
DATA    EQU   10H
STS     EQU   11H
CMND    EQU   11H
; STATUS BYTES
WCD     EQU   0F0H
DB1STS  EQU   80H
DB2STS  EQU   90H
DB3STS  EQU   0A0H
DB4STS  EQU   0B0H
; COMMANDS
SWCMND  EQU   25H
MWCMND  EQU   0CH
; BUFFERS
CMDBUF  DB    0F005H
; LONG ENOUGH FOR ANY COMMAND
START:  LXI   H, CMDBUF
        MVI   M,0          ;LOAD TOTAL NUMBER OF
        INX   H            ;BYTES TO BE SENT
        MVI   M,32
        INX   H
        MVI   M, MWCMND    ;LOAD COMMAND BYTE
        INX   H
        MVI   M,10         ;NUMBER OF DATA BYTES
        MVI   D,10H        ;ASSUME WE ARE
        MVI   E,0          ;WRITING TO EVERY 10TH
        MVI   C,0          ;ADDRESS STARTING AT 1000H
                           ;ASSUME WE ARE USING THESE
                           ;LOCATIONS FOR ERROR LOGGING
                           ;AND ARE NOW RESETTING THEM TO
                           ;ZEROS.
LDBFLP: INX   H
        MOV   M,D
        INX   H
        MOV   M,E
        INX   H
        MVI   M,0
        INR   C
        MOV   A,C          ;HAVE LOADED ALL
        CPI   10H          ;TEN BYTES, CAN NOW
        JZ    SNDERR       ;SEND COMMAND
        MOV   A,E
        ADI   10
        MOV   E,A
        JMP   LDBFLP
SNDERR: CALL  SNDCMD
        RET
```

AFN-02020A

Figure 7. 8298 Multiple Write Routine

SYSTEM CONFIGURATIONS

The two E^{2}PROM-8298 configurations presented in this section give the systems designer an option of minimal overhead hardware versus minimum E^{2}PROM access time. Each is described in detail including a discussion of the relative advantages and disadvantages to the user.

Indirect Configuration

The indirect configuration is so named because the host accesses the E^{2}PROMs indirectly—the 8298 acts as a buffer in all transmissions between the host and the E^{2}PROMs. In this hardware implementation, all write and read operations must be executed from the 8298, making full use of its input and output capabilities.

The effect of going through the 8298 is to slow the retrieval of data from the E^{2}PROMs. The host cannot read the data directly at system speed from the E^{2}PROM. A general example of a read operation would be as follows: The host first sends the 8298 a read command. Upon receiving the command, the 8298 decodes it, reads in the data from the E^{2}PROM and places it in the data-out register. The host can now retrieve this data. For write cycles, there is little performance degradation effect from using the Indirect Configuration since the 8298 will oversee a write operation in any case.

A major advantage of the Indirect Configuration is the ease of implementation. The hardware is relatively simple and straightforward, as is the interface to the host. All that is required for the interface is access to the two adjacent I/O ports within the 8298. This would make it simple to add an 8298 to an existing system, since no changes need to be made to the host's memory map and interfacing is relatively simple.

Indirect Configuration Hardware

Indirect configuration circuitry beyond the 8298 and 8243 described earlier includes an address decoder, programming pulse generation circuitry, and chip erase pulse generation circuitry. The schematics for this hardware are shown in Appendix C.

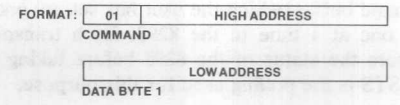
The address decoder is the Intel 8205. It selects via address lines 11, 12 and 13 the appropriate device in the E^{2}PROM array. Programming and Chip Erase pulse generation circuitry are described in Ap-102 available from your Intel representative.

Indirect Configuration Commands

Though the hardware implementation of the Indirect Configuration is relatively simple, there are numerous commands available that make the 8298 versatile to use. The following discussion covers all commands that may be used with this configuration.

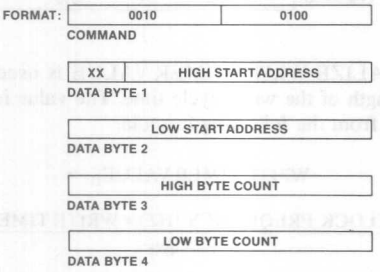
READ COMMANDS

There are two read commands available to send to the 8298. Both return the data to the data-out register of the 8298. INDIRECT READ is used to read a single byte of data from an E^{2}PROM. Note that the command and high address are combined in one byte.



AFN-02020A

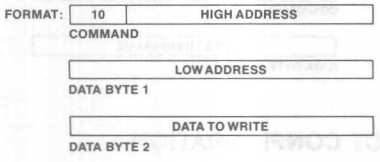
SERIES READ is used to read any number of sequential data bytes up to 16K bytes. The command consists of five bytes, so that it is usually more efficient to use SERIES READ for reading 3 or more bytes.



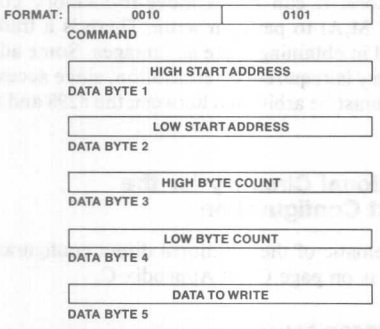
WRITE COMMANDS

There are three commands for writing to the E^2PROM via the 8298. They each have functions to make them useful for different applications.

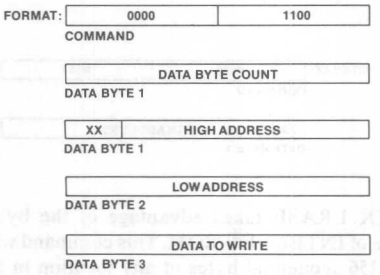
INDIRECT WRITE is similar in format to INDIRECT READ; it is for writing a single byte.



SERIES WRITE is the counterpart to SERIES READ. Note that once the command and data bytes have been sent, the 8298 asks for data byte 1 until "count" number of bytes have been sent. Up to 32 bytes of data are buffered in RAM by the 8298. Thus the data can be sent in a "burst."



MULTIPLE WRITE has no counterpart READ command. It is similar to INDIRECT WRITE in that it may write to nonsequential locations. The data and addresses are buffered in the 8298 so that up to 9 data/address groups may be sent in a burst fashion (i.e., no waiting for erase/write cycle).



The write commands have been optimized to increase throughput as much as possible. One example is the buffering of data (and possibly address) for SERIES WRITE and MULTIPLE WRITE. Another way is by reading first to check data at the address to be written . Under certain conditions, the byte erase can be avoided and thus 10 msec can be saved. If the data is identical with that to be written, the write can also be avoided. See Figure 8 below for the flowchart of the erase/write checking routine.

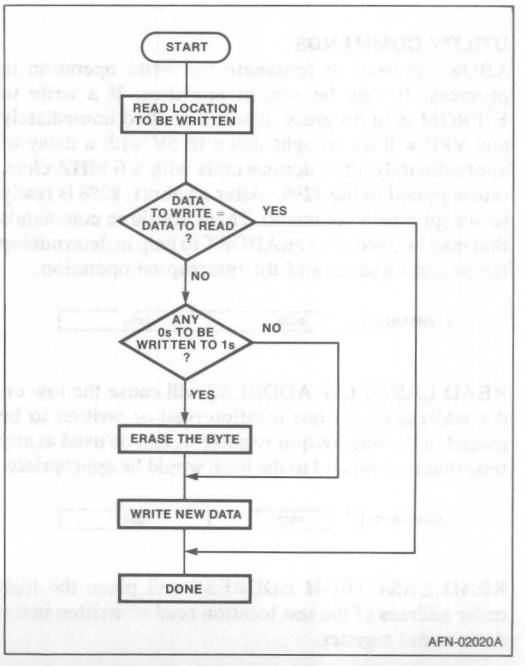
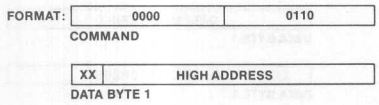


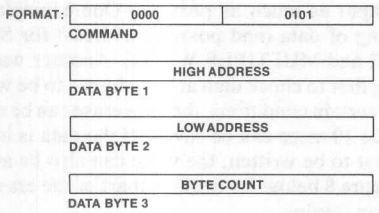
Figure 8. Erase/Write Flowchart

ERASE COMMANDS

There are two erase commands. CHIP ERASE will erase one complete E^2PROM. The chip to be erased is specified by sending the upper six address bits of the chip.

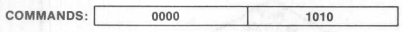


BLOCK ERASE takes advantage of the byte erase feature of INTEL's E^2PROM. This command will erase up to 256 sequential bytes at any location in the 16K array.

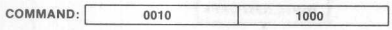


UTILITY COMMANDS

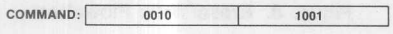
ABORT is used to terminate the write operation in progress. It may be sent at any time. If a write to E^2PROM is in progress, it will be halted immediately and VPP will be brought down to 5V with a delay of approximately 100 microseconds with a 6 MHZ clock rate supplied to the 8298. After an abort, 8298 is ready to accept a new command. There are three commands that may be used after an ABORT to help in determining the address and data of the interrupted operation.



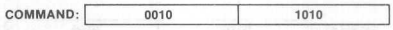
READ LAST LOW ADDRESS will cause the low order address of the last location read or written to be placed in the data output register. It can be used at any time that a command to the 8298 would be appropriate.



READ LAST HIGH ADDRESS will place the high order address of the last location read or written in the data output register.



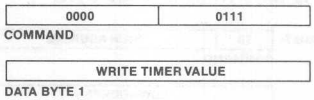
READ LAST WRITE DATA will put the data written to the E^2PROM during the last write cycle into the data output register.



INITIALIZE WRITE TIMER VALUE is used to set the length of the write cycle time. The value is determined from the following formula:

$$\text{WRITE TIMER VALUE}_{10} = 256 - \frac{(\text{CLOCK FREQUENCY (HZ)} \star \text{WRITE TIME (SEC)})}{480}$$

For example, with a 10 msec write time and a 6 MHz clock the timer value should be 131_{10}. Note: This is the default value (131_{10}). It is recommended that the 8298 be run at 6 MHz to achieve maximum efficiency. If it is run at a slower rate, the required value will have to be sent every time a reset is issued and on power-up.



DIRECT CONFIGURATION

Advantages/Disadvantages

The Direct Configuration allows the CPU to bypass the 8298 to read the E^2PROMS, thus taking advantage of the fast access times of the E^2PROMS. Write operations do not bypass the 8298, but the host can address the E^2PROM directly instead of through the I/O port. Therefore, it can use a move-to-memory command (MOV M,A) to perform write. There is a tradeoff involved in obtaining these advantages. Some additional circuitry is required for arbitration, since access to the array must be arbitrated between the 8298 and the host CPU.

Additional Circuitry for the Direct Configuration

A schematic of the additional direct configuration circuitry is on page C5 of Appendix C.

Host/8298 Interface

The major additions to the circuitry of the Indirect Configurations are the arbitration circuitry, and circuitry to force a write to the data-in register of the 8298

when a direct write is being made to E^2PROM address space. The 8298 recognizes this write as a direct write, gets the address from the additional address latches and completes the write. The arbitration circuitry consists of a D flip flop clocked by the SYNC output of the 8298. The SYNC output occurs once every time the 8298 performs an instruction. The 8298 sends out an EEREQ to see if it can access E^2PROMS. If it can, as determined by the arbitration circuitry, EEACK will sense a high level and the 8298 will continue the command. There are also latches for address retention and a transceiver for data in this configuration.

The addresses for direct reads or writes are strobed into these address latches which have outputs tied directly to the E^2PROM address pins. On a direct read or write the tristate outputs of these latches are enabled. Data can be read direct from the E^2PROM array using the data transceiver. The data being returned from a read operation goes through the transceiver when enabled by a read signal ($\overline{RD}$) from the host. All writes continue to be supervised by the 8298, so the host *must* read the status register of the 8298 before writing to the E^2PROMs to make sure that the 8298 is waiting for command.

Direct Configuration Commands

The added capabilities for the Direct Configuration are ease and speed of access for the host rather than a greatly extended command set. In fact, there are only two additional commands for the Direct Configuration.

ENABLE DIRECT WRITE is sent to inform the 8298 that the host may wish to use the "direct write." If this command isn't sent, and the host tries to do a direct write, it will be treated as an illegal command. If the host processor will be using direct writes, the ENABLE DIRECT WRITE command must be sent before attempting a direct write.

COMMAND:	0010	0010

DISABLE DIRECT WRITE signals the 8298 that the host no longer wishes to use the direct write utility.

COMMAND:	0010	0011

NOTE:
The 8298 defaults to this condition (on reset and power-up).

APPENDIX A
COMMAND SUMMARY

Table A summarizes the commands available to the host processor. It includes DIRECT READ and WRITE for completeness. The first column indicates the mnemonics for the command. The second column indicates the number of bytes in the command. The third column indicates the byte sequence, and the fourth shows the format of the byte. The last column shows the status which will be displayed in the 8298 status register when it is done processing the byte and ready to continue. The status mnemonics are defined in Table A.2.

Table A.1

Command	# of Bytes for Commands	Byte#	Format	Next Status
INDIRECT READ	2	1	01 High Address	DB1
		2	Low Address	OBF
SERIES READ	5	1	0010 0100	DB1
		2	XX High Start Address	DB2
		3	Low Start Address	DB3
		4	High Byte Count	DB4
		5	Low Byte Count	OBF
INDIRECT WRITE	2+ WRITE DATA	1	01 High Address	DB1
		2	Low Address	DB2
		3	DATA TO WRITE	WCD
SERIES WRITE	5+ WRITE DATA	1	0010 0101	DB1
		2	XX High Start Address	DB2
		3	Low Start	DB3
		4	High Byte Count	DB4
		5	Low Byte Count	DB1
			DATA TO WRITE	DB1/WCD NOTE 1
MULTIPLE WRITE	2+ ADDRESS/WRITE DATA	1	0000 1100	DB1
		2	Byte Count	DB1
		*	XX High Address	DB2
		*	Low Address	DB3
		*	DATA TO WRITE	DB1/WCD NOTE 2
BLOCK ERASE	4	1	0000 0101	DB1
		2	High Start Address	DB2
		3	Low Start Address	DB3
		4	Byte Count	WCD
CHIP ERASE	2	1	0000 0110	DB1
		2	XX High Address	WCD NOTE 3
ENABLE DIRECT WRITE	1	1	0010 0010	WCD
DISABLE DIRECT WRITE	1	1	0010 0011	WCD
ABORT	1	1	0000 1010	WCD
READ LAST LOW ADDRESS	1	1	0010 1000	WCD

Table A.1. Continued

Command	# of Bytes for Commands	Byte#	Format	Next Status
READ LAST HIGH ADDRESS	1	1	0010 1001	WCD
INITIALIZE WRITE TIMER VALUE	2	1 2	0000 0111 Write Timer Value	DB1 WCD

*ADDRESS + WRITE DATA

NOTES:
1. SERIES WRITE returns a status of DB1 whenever it is waiting for write data. When BYTE COUNT bytes have been sent, it will return a status of WCD. See SERIES WRITE command, under Indirect Configuration Commands, for more information.
2. MULTIPLE WRITE works in a loop format once the command and byte count are received. It requires groups of three bytes and cycles through the status as: -DB1—DB2—DB3- until "count" groups of bytes are received. It then returns a status of WCD.
3. High address is the high-order six bits of the highest address in an individual E^2PROM. An example might be: FOR 7FFH as the highest address in an E^2PROM, to erase would require sending 0000 0111 as the high address.

Table A.2. Status Bit Representations

Status Mnemonic	Binary Representation	Hex Representation
WCD	1111 0000	F0
DB1	1000 0000	80
DB2	1001 0000	90
DB3	1010 0000	A0
DB4	1011 0000	B0
OBF	0000 0001	01

NOTE:
If *DIRECT WRITES ARE ENABLED*, bit 2 will be a one (e.g., WCD would be 1111 0100B or F4H).

APPENDIX B
CONFIGURATION COMMAND SUMMARY

This Appendix contains a listing of the commands available to the 8298 E^2PROM controller in each of the configurations. It has been provided so that the user can easily look up available commands for their 8298 application.

Table B.1. Configuration and Applicable Commands

Commands	Indirect	Direct
Indirect Read	X	X
Series Read	X	X
Indirect Write	X	X
Series Write	X	X
Multiple Write	X	X
Chip Erase	X	X
Block Erase	X	X
Abort	X	X
Read Last Low Address	X	X
Read Last High Address	X	X
Read Last Write Data	X	X
Initialize Write Timer Value	X	X
Enable Direct Write		X
Disable Direct Write		X
Direct Write		X
Direct Read		X

APPENDIX C
HARDWARE SCHEMATICS

Mnemonic	Description	Origin/Schematic
A0–A13	System address bus	Host/C1, C5
D0–D7	System Data bus	Host/C1, C5
$\overline{\text{Reset}}$	System reset, active low	Host/C1, C3, C6
Reset	System reset, active high	— /C1
$\overline{\text{RD}}$	Host read, active low	Host/C1, C5
$\overline{\text{WR}}$	Host write, active low	Host/C1, C6
IO/$\overline{\text{M}}$	Host signal	Host/C1, C6
IA0–IA13	Internal address bus	8298 & 8243/C2, C5
ID0–ID7	Internal data bus	8298/C2, C5
EEN	E^2PROM enable, active high	8243/C3, C5, C6
$\overline{\text{EEN}}$	E^2PROM enable, active low	— /C2, C6
EEREQ	E^2PROM request	8243/C6
EEACK	E^2PROM acknowledge	8298(I)/C6
$\overline{\text{VPPEN}}$	V_{PP} circuitry enable, active low	8243/C4
$\overline{\text{CRD}}$	8298 read signal, active low	8243/C3
ALEQ	ALE (Qualified)	C6
HOSTACK	Host Acknowledge	C6/C5
$\overline{\text{HOSTRD}}$	Host Read of 2816s	C5
$\overline{\text{IBF}}$	Input Buffer (8298) not full, active low	8298/C1/C6
OBF	Output Buffer Pull	C1/C6
CS1, CS2	Chip select to enable 8298	C6
CERASE(ID7)	Chip erase signal	C3

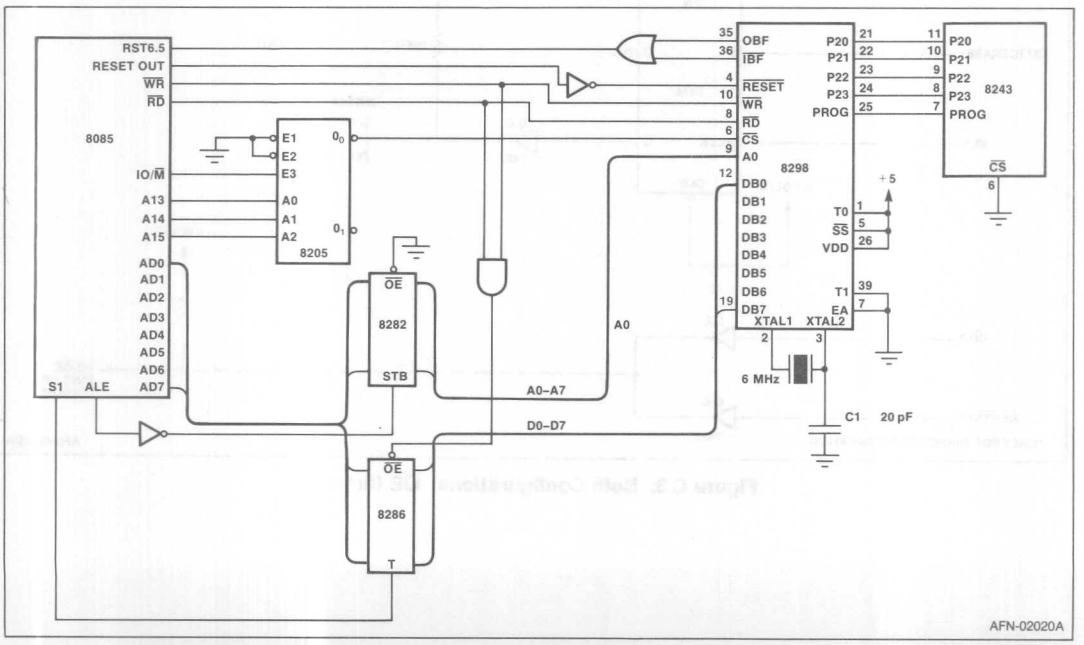


Figure C.1. Both Configurations: Host CPU/8298 Interface Example Using 8085

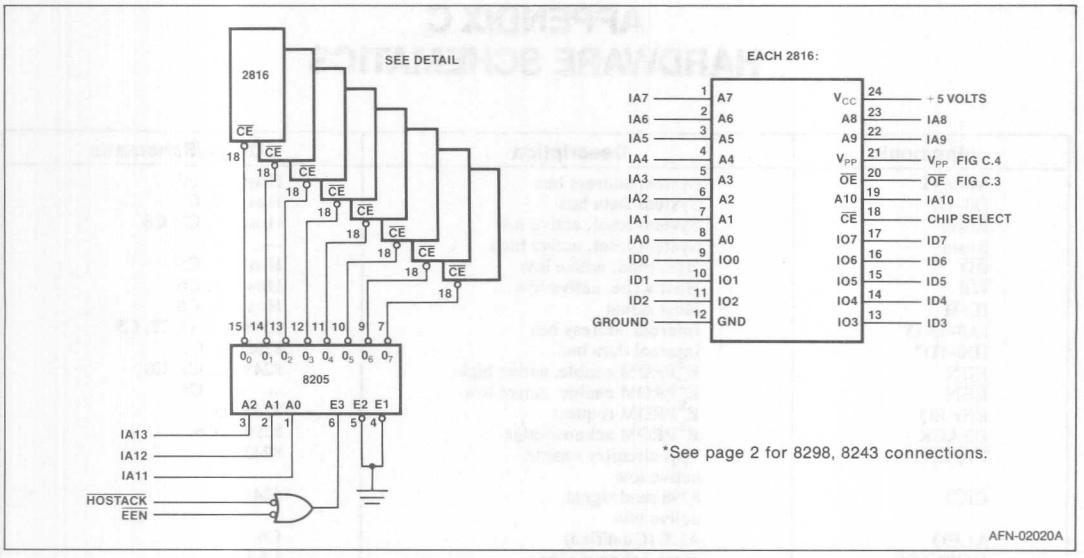


Figure C.2. Configurations: E^2 Array—Eight (8) 2816

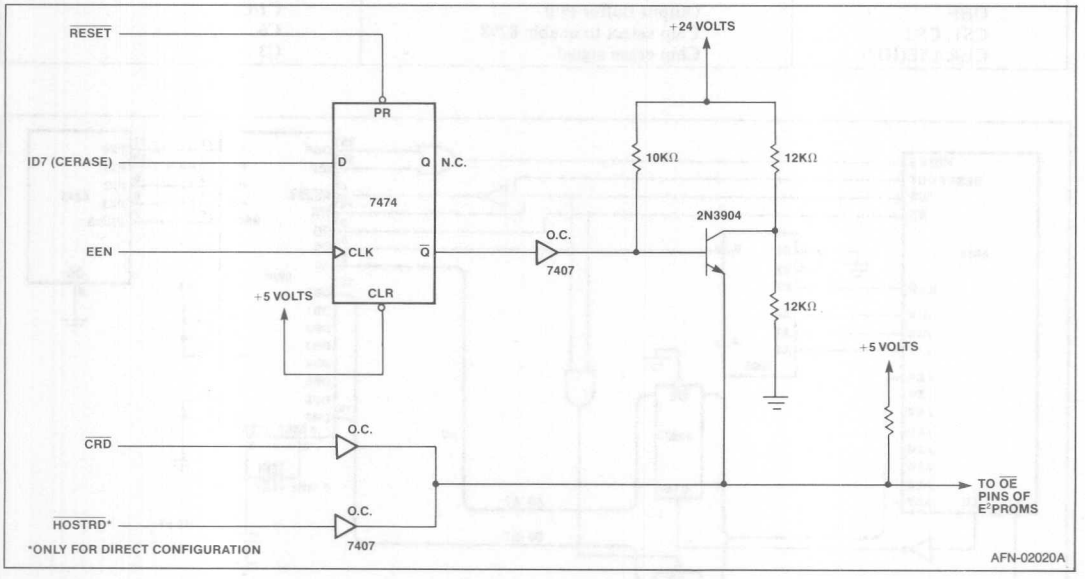


Figure C.3. Both Configurations: $\overline{OE}$ Circuitry

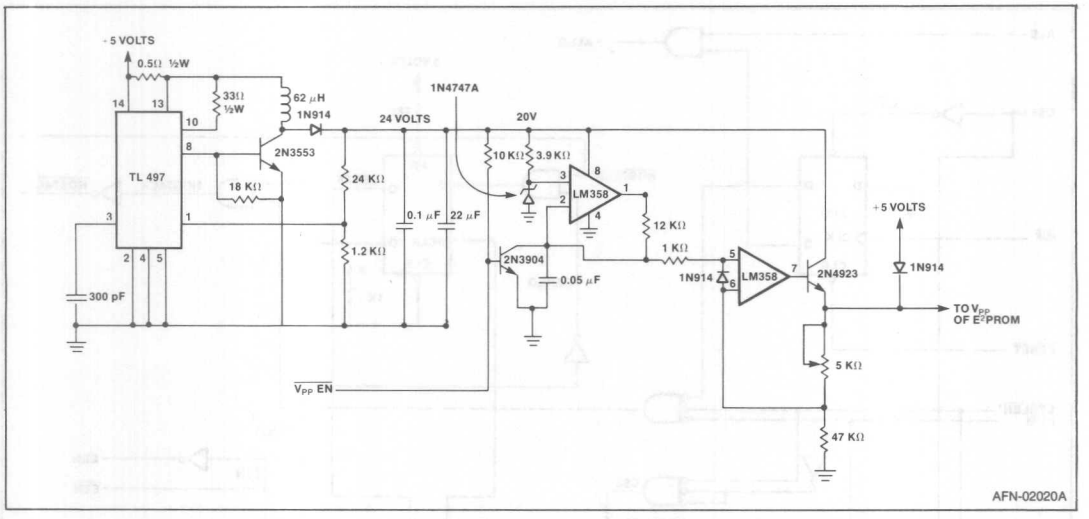


Figure C.4. Both Configurations: V_{PP} Circuitry (5 Volt Only—DC to DC Converter)

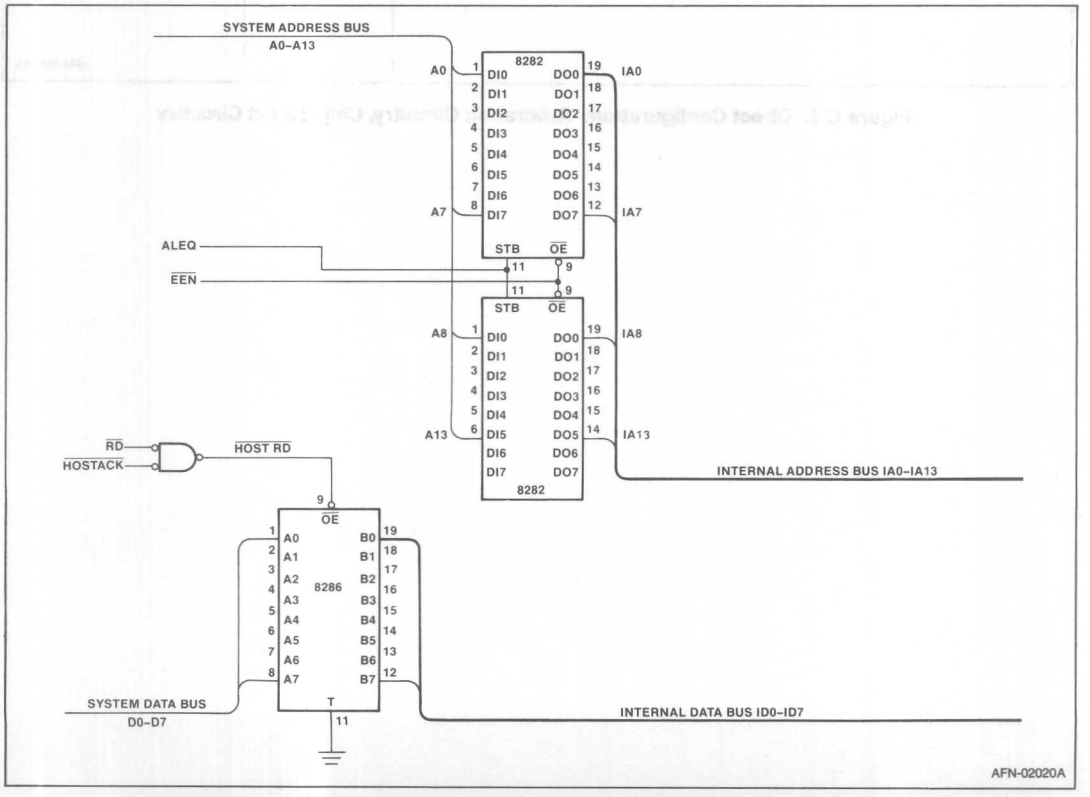


Figure C.5. Direct Configuration: Data/Address—Latches/Drivers

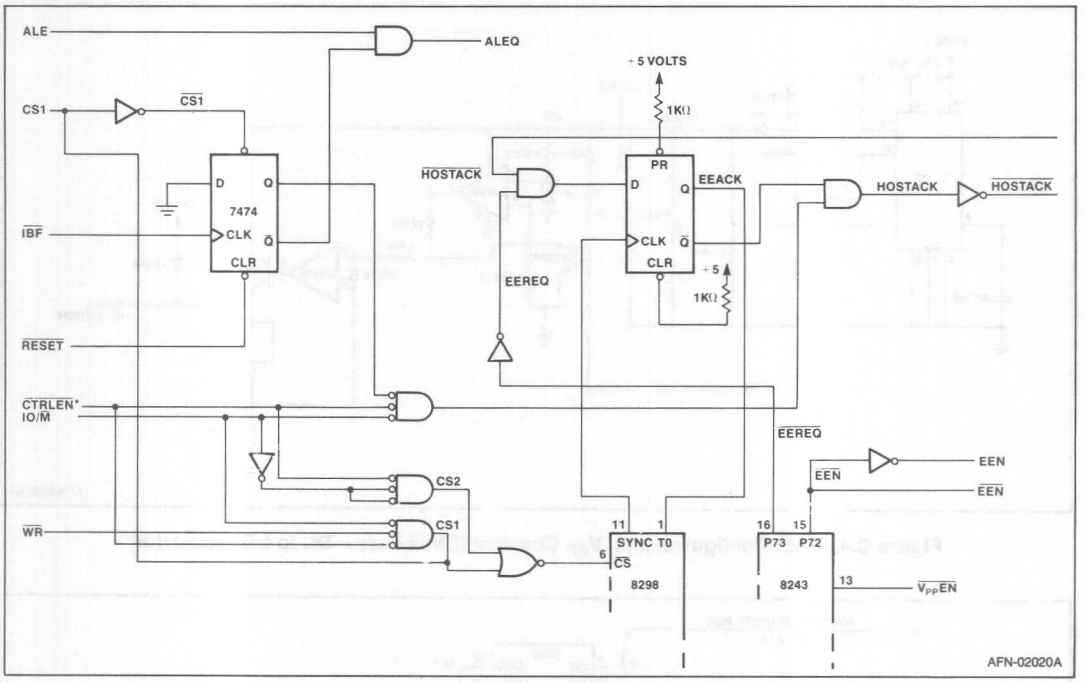


Figure C.6. Direct Configuration: Arbitration Circuitry, Chip Select Circuitry